ALL ABOUT

AMOS

VOLUME 1 THE ALTERNATIVE AMOS RESOURCE ISSUE 2

LOADS OF PROGRAMMING TUTORIALS AND IN-DEPTH ARTICLES

PRODUCTION SECRETS OF EUROPRESS SOFTWARE REVEALED

GREAT NEW LOOK SECOND ISSUE!!!

CONTENTS

CREDITS

The contributors to this issue who deserve a big thank you and a juicy mince pie are:

Colin White
Gary Symons
Richard Vanner
Richard Gale
Leo Douglas
Kyle Rees

EDITOR'S REVENGE

HE'S BACK

I am writing this Editorial whilst sipping some British Rail coffee on my way up to Macclesfield, where I hope to finish off the fabled CDTV Fun School 3 series of programs. Before you ask - NO! I don't have a portable computer, this is all being written on white A4 with a HB pencil!! Cheap, dependable and it doesn't weigh 20 kilos.

IT'S A SUCCESS!!

The first issue of "ALL ABOUT AMOS" was a great success. The input you have provided has allowed us to focus the contents of the magazine onto the areas which YOU want to see explored. Remember, this is your magazine and only you can influence its contents.

Our thanks must go to all of those people who took the time and trouble to return the readers survey, your responses are helping us to improve this magazine for the better. The most requested article was "HOW TO WRITE A WHOLE GAME", a tricky one due to the fact that we don't have a cover disk. The question is, if we ran such a series of articles and published a disk containing all of the graphics etc. how many of you would actually buy it? If you like the idea or have a better suggestion, drop us a line.

As Publisher & Editor I was hoping to squeeze a few extra pages into this issue, but although subscriptions have been reasonable they are not enough to allow expansion. If you have any friends or relatives who read your copy of "ALL ABOUT AMOS" get them to subscribe, it will benefit you in the long run.

CONTINUED ➡

ALL correspondence for "ALL ABOUT AMOS" should be addressed to:
PETER HICKMAN, "ALL ABOUT AMOS", 36 CLEVERLY ESTATE, WORMHOLT ROAD
LONDON W12 0LX, ENGLAND
Photographs, pictures and other materials submitted for publication cannot be returned unless accompanied by a stamped self-addressed envelope.
The publisher (P.J.HICKMAN) cannot accept responsibility for correspondence lost or damaged during transit.
Unless otherwise stated all material contained in this issue is © Copyright 1991 P.J.HICKMAN.
If you have any subscription enquiries please phone 081 749 6086 during OFFICE HOURS ONLY please.

This issue is dedicated to the late Freddie Mercury, for all the hours of joy and entertainment he gave not only us, but the whole world.

THE COMMODORE SHOW

Well, it was nice to see so many of you at the recent World of Commodore Show. I hope you enjoyed your time there (crowded wasn't it?). As you can see by the cover, which by the way, shows François Lionet's reaction to Richard Vanner suggesting that AMOS should be ported over to the NINTENDO GAMEBOY, a good time was had by all (a stereotypical magazine-type phrase that means the drinks flowed smoothly and the after-show parties were better than a night watching Sky Movies).

THIS ISSUE

We have introduced a couple more types of features in this issue of the magazine, the most noticeable being the "HANDY HINTS" column. We feel these type of boxed off programs and hints provide a structure to the magazine which makes it easier to read and follow. Do you agree? If not write and tell us and it will be changed!!

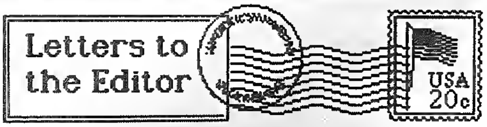
AND FINALLY.....

On behalf of everybody involved with "ALL ABOUT AMOS" I would like to take this opportunity to wish all of our readers a very merry Christmas and a prosperous, happy new year. See you in Feburary.

Peter Hickman
(*Editor*)

A PLEA FOR POST!

We want an AMOS problem page and letters page for the next issue. So if you have trouble using AMOS, whether it is a programming matter or not, write to us at the usual address and we will try our very best to sort it out. If you send in a disk with an example of the problem please remember to enclose a stamped self-addressed envelope. I promise that all of your disks will be sent back, but without an S.A.E. it may take quite a while!

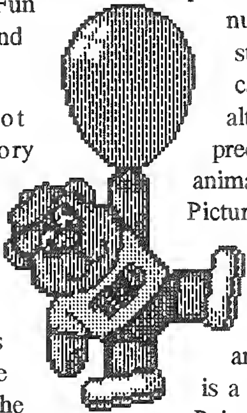


TEDDY BEARS ALL

BY COLIN WHITE

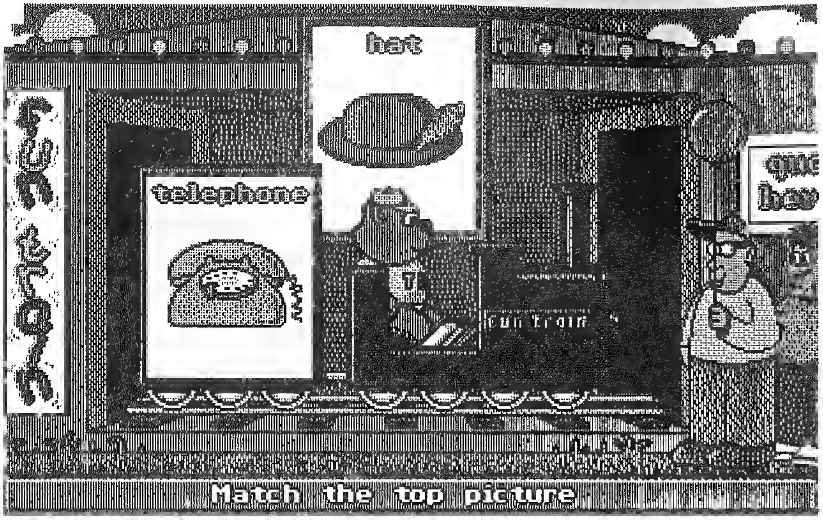
FUN SCHOOL 4 is the follow-up to the highly successful Fun School 3 series and contains a set of six little proggies for your little lad or lass to play with, enjoy and learn from. Obviously, being intended for very small children, these programs have a greater emphasis on large, colourful graphics and gamey elements rather than containing the heavier educational content which can be found in the other two Fun School 4 titles, 5 to 7 years and Over 7s.

ADDITION This is, not surprisingly, an introductory program to addition and simple sums, using familiar objects. The game is set inside a classroom in Fun School where Mr Ted the teacher is keeping his students behind until they can answer the sums that he sets for them. The player must help the pupils by answering the sums, and as each one is correctly answered, one of the pupils is allowed to get up and leave the classroom. When all the students have gone Mr. Ted also leaves the room, giving a friendly wave as he goes. While playtesting this one I did notice an unfortunate repetition of one of the sums. Eight tanks were displayed twice in one game on the first level, which may be a little confusing to a small child who might think "Hang about, I've already answered this one. Obviously the programmers have failed to implement an adequate routine to prevent repetition occurring within the same level". Or maybe not. You get the idea, anyway.



TEDDY PAINT A graphics package which your little kiddy can use to practice and perfect his or her artistic talents by building up pictures using a combination of hand drawing and predefined pictures. There are a number of complete drawings stored on disk which the child can load and play about with, or alternatively he or she can use the predefined pictures of objects and animals to build their own scenes. Pictures to choose from include a beach scene and a country picnic scene.

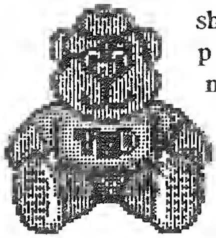
FUN TRAIN Teaches object and word recognition. Fun Train is a variant of the familiar Matching Pairs theme. The Fun School Teddy has taken a job at a fairground running the Fun Train, a locomotive which pulls an object in a truck behind it. The player is shown a picture of an object (such as an ice cream, for example) and must press a key when Ted's train emerges from the tunnel carrying the picture card which matches it. At the end of each level, Ted jumps down off the train and gives a congratulatory wave to the user, while the lights over the ride flash on and off. Ted and the picture cards are very attractive and the surrounding scene, although not quite of the same



"FUN TRAIN APOLOGISES FOR THE DELAY TO ITS SERVICE CAUSED BY TEDDIES ON THE LINE AT MACCLESFIELD...."

standard, still has cartoony appeal.

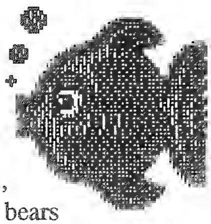
TEDDY'S HOUSE Teaches colour identification and object recognition. On the first level of Teddy's House, the screen displays a picture of a house, garage, garden fence, gate and post box, all in black and white which the child can colour using one of 8 colours shown along the bottom of the screen. The user is asked what colour he or she would like to paint each object and if there is enough paint of the desired colour left (see comment below) the appropriate part of the picture is filled in with the required colour. A comment on the painting of objects - if the player tries to paint too many parts of the picture in the same colour, the paint runs out. This adds a touch of realism to the process, but can be annoying if you are trying to colour co-ordinate the house, garage and fence! On the upper levels of the game, a coloured version of the scene is shown and the player is asked to identify the parts of the picture and the colours used. As you progress up the levels in this game a bird appears sitting on the fence, until on the third and final level, three of these curious birds are shuffling about and peering around. An interesting animated cartoony addition that gives extra appeal. Finally, a little naffness warning: Because of the random choice of colour combinations for the scene on the third level, you may need your Anti-Tastelessness shades on when you play, to protect you from the clashing colours! One technical point to the programmers - If the player tries to paint an object he or she is not allowed to, such as the sky, a message pops up saying "You cannot paint something that doesn't exist". Perhaps more thought



should've gone into phrasing this message, because if it's visible on the screen, as far as the child is concerned, it DOES exist - whether it can be painted or not.

TEDDY'S KARAOKE Aaarghhh! Nooooo! Sadly, Yes. The dreaded karaoke craze has even struck the Europress product range but don't worry (too much) - the songs are traditional nursery rhymes such as "Hey, Diddle, Diddle" and "Jack and Jill", which MC Teddy D (yes, really!) "spins" on his "decks" (or something like that). The lyrics to each nursery rhyme are displayed as it is played and a little jumping ball steps through each line of the rhyme in time to the music. While the tunes play, colourful disco lights flash and our "street tuff" DJ "bosses" the turntables for all he's worth. An interesting combination of classic nursery rhymes and modern music presentation which might seem to be as musically deficient as "Gazza sings Nat King Cole", but it really is worth a look. Honest.

TEDDY'S BOOKS Teddy's Books covers number estimates and deduction by elimination. The player is shown a cosy bedroom scene with Daddy bear reading a storybook from a selection of books on the shelf over the bed, to his six kiddies as they settle down for a good night's sleep. Unfortunately, Daddy bear can't remember which numbered book he is reading, so the player needs to work out which number it is and tell him. As each number is correctly guessed, one of the Baby bears turns over and goes to sleep, and when all the bears are asleep, Daddy bear gets up and leaves the room, turning the light out as he goes. Although the whole game is well presented graphically, probably the best extra touch added to the graphics in Teddy's Books is the way the stars twinkle outside the bedroom window. Finishing touches like this show care has been taken to make the end product as attractive as possible.



Overall, the educational content of FUN SCHOOL 4 for the Under 5s is rather thin, but its weediness is made up for by the sheer "Pass the sickbag" cuteness of the cartoony graphics, and as with the birds on the fence in Teddy's House for example, it is features like this which are especially important in very small childrens' games because they add a greater depth of interest and a desire to learn. The reward sequences when the child answers a question accurately are also pleasing and likely to entertain. On all the games, animation for hopping between levels is good, but the mouse pointer movement is slightly sluggish on the main menu. Spreading the Under 5s programs over two disks may be a little inconvenient when small children are the target users, but at least the main menu is provided on both disks which reduces the nuisance of disk swapping. Otherwise, both presentation and marketing of all the FUN SCHOOL 4 packages is excellent. A free sticker is given with every box, which is a useful feature, but perhaps Europress Software should have distributed badges as they did with Fun School 3, or replaced the "Belt Up In The Back" car window sticker provided in the Under 5s box with an ordinary sticker. After all, not every family owns a vehicle, but anyone can wear a badge or use an ordinary sticker.

THE REVIEWER'S QUALIFICATIONS

Colin White was once a schoolchild and has spent at least 60 months of his life under 5 years of age.

Ed's Comments

The Fun School series of programs has topped the charts on most computer formats for some time now, and deservedly so!

All other educational software pales in comparison with this stuff, in fact the only program which partially stands up to Fun School 4 is Fun School 3 (and FS4 offers better graphics, sound and long term appeal compared to that!).

If you are looking for a good piece of educational software for the youngsters, buy Fun School 4. It offers fantastic value for money and you certainly won't be disappointed.

ALL ABOUT FUN SCHOOL 4

TITLE: Fun School 4 for the Under 5s.

PUBLISHER: Europress Software.

REQUIRES: Standard Amiga 500 (also works on new A500+). Mouse and keyboard. Will run in $^1/_2$Mb, but should ideally be run with 1Mb or more.

PRICE: £24.99

Available from:

All major software stockists

HANDY HINTS

WARNING!!! There are two AMOS commands which eat up your memory and don't give it back. This is due to a slight problem with the Amiga (so I am told!!), so unfortunately there is no solution. The commands are

```
GET FONTS or GET DISC FONTS
```

and

```
DIR$=
```

Both of these commands are okay if you use them once, but any more than that and your memory will start to disappear!

THE 3D ANGLE

An on-going comprehensive guide to 3D graphics generation using AMOS

BY RICHARD GALE

If you have seen games like Elite and Starglider then you will have realised that the main graphics were not like ordinary sprites. These graphics are constructed using lines and points, which can be manipulated mathematically. This style of graphics is called 3D graphics.
In this article I will be covering vector graphics, graphics which are made up of only straight lines.
In 3D graphics there are three axis:-
X axis - from the left to the right
Y axis - from the top to the bottom
Z axis - from the front to the back
(into the monitor screen!)
The first thing which you must do is create a 3D image, and for this example I will use the simplest object, a cube. See figure 1.
This cube has the points shown below the diagram in figure I.
Once the points have been defined we have to declare where lines have to be drawn between these points.
This gives the lines :-

Line	Start	End
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	4	8
11	3	7
12	2	6

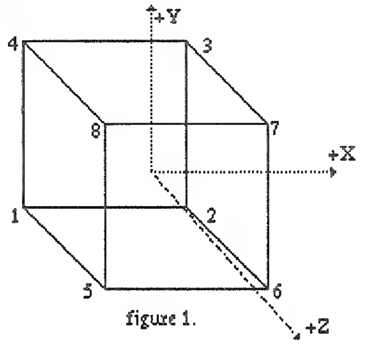


figure 1.

Point	X	Y	Z
1	-20	-20	-20
2	20	-20	-20
3	20	20	-20
4	-20	20	-20
5	-20	-20	20
6	20	-20	20
7	20	20	20
8	-20	20	20

Now we have the image defined we need to be able to move it...

3D TRANSFORMATIONS

There are three types of 3D transformations, these are translations, rotations, and scaling.

TRANSLATIONS

Translations are used to move the image in the 3D space, the object can be moved up/down, left/right, nearer/further, ie along one or more axis. Three variables are used to represent the translation on each axis :-
TRANX for translations on the x axis

Translations can be applied using the following matrix :-
(XNEW, YNEW, ZNEW, 1)=(XOLD+TRANX, YOLD-TRANX, ZOLD-TRANZ, 1)
or the following formulae :-
XNEW = XOLD + TRANX : YNEW = YOLD - TRANY : ZNEW = ZOLD - TRANZ
Note: the last two formulae are subtractions so that positive y is up, and positive z is nearer.

TRANY for translations on the y axis
TRANZ for translations on the z axis.
For example :-
positive TRANX = move the image to the right.
negative TRANX = move the image to the left
positive TRANY = move the image upwards.
positive TRANZ = move the image nearer to the user.

ROTATIONS

Rotations are used to change the image's aspect, how much the object has been spun in each of the three axes. There are three variables to control rotations, these are :-
ROTX - rotation in degrees about x axis.
ROTY - rotation in degrees about y axis.
ROTZ - rotation in degrees about z axis.
Rotation of ROTX degrees about the X axis:
XNEW = XOLD
YNEW = YOLD * cos(ROTX) + ZOLD * sin(ROTX)
ZNEW = ZOLD * cos(ROTX) - YOLD * sin(ROTX)
Rotation of ROTY degrees about the Y axis:
XNEW = XOLD * cos(ROTY) - ZOLD * sin(ROTY)
YNEW = YOLD
ZNEW = XOLD * sin(ROTY) + ZOLD * cos(ROTY)
Rotation of ROTZ degrees about the Z axis:
XNEW = XOLD * cos(ROTZ) - YOLD * sin(ROTZ)
YNEW = XOLD * sin(ROTZ) + YOLD * cos(ROTZ)
ZNEW = ZOLD

SCALING

Scaling changes the size of the image making the object larger or smaller on one or more axis. Three variables are used to scale an object, they are :-
XSCALE# scale factor for x axis
YSCALE# scale factor for y axis
ZSCALE# scale factor for z axis
Scaling can be performed by the following formulae :-
XNEW = XOLD * XSCALE#
YNEW = YOLD * YSCALE#
ZNEW = ZOLD * ZSCALE#
Note: XOLD, YOLD, and ZOLD contain the old position of the point. XNEW, YNEW and ZNEW hold the new calculated position of the point.

MULTIPLE CALCULATIONS

To perform multiple 3D calculations all that is required is to execute each of the transformations one after another.
For example to transform, then rotate about the x and y axes :-
Do transformation:-
XNEW = XOLD + TRANX
YNEW = YOLD - TRANY
ZNEW = ZOLD - TRANZ
Copy new co-ords into XOLD, etcetera, ready for next calculation:-
XOLD = XNEW
YOLD = YNEW
ZOLD = ZNEW
Do x rotation:-
XNEW = XOLD
YNEW = YOLD * cos(ROTX) + ZOLD

* sin(ROTX)
ZNEW = ZOLD * cos(ROTX) - YOLD * sin(ROTX)
XOLD = XNEW
YOLD = YNEW
ZOLD = ZNEW
Do y rotation:-
XNEW = XOLD * cos(ROTY) - ZOLD * sin(ROTY)
YNEW = YOLD
ZNEW = XOLD * sin(ROTY) + ZOLD * cos(ROTY)

PERSPECTIVE

Finally, after all calculations, we must display our 3D image. To display objects with any degree of realism some perspective techniques must be used. In conventional drawings vanishing points are used to get the correct positioning. As true 3 dimensional effects cannot be easily achieved on a monitor screen, instead a trick is used to modify the x and y co-ordinates to give the impression of depth. Points near to the screen are spaced apart, whereas points in the distance are placed closer together.

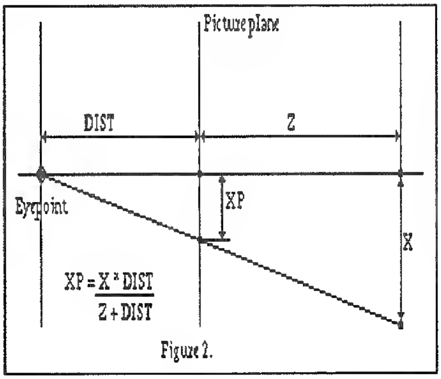


Figure 2.

To calculate the correct positioning the following matrix must be applied to each point :-
(XNEW, YNEW, ZNEW, 1)=(XOLD * F, YOLD * F, 0, 1) where F is calculated as :-

$$F = \frac{DIST}{ZOLD + DIST}$$

The perspective translations from a 3D space image onto the 2D monitor screen image are :-
X-coord = X * DIST / (Z + DIST)
Y-coord = Y * DIST / (Z + DIST)
where X,Y and Z are the positions in the three planes, and DIST is the distance between the screen and your eyes. (all are measured in pixels). See figure 2.
Well, that's the theory over with, now for the implementation.

IMPLEMENTATION

To implement the 3D routines the following steps must be followed :-
Main program:
1) Initialize 3D image. The point data is stored in three arrays X(), Y() and Z().
2) Initialize line data. The line data is stored in two arrays LINEA() and LINEB().
3) Initialize other variables:
DIST - distance to eye point
XCEN, YCEN - centre of screen
TRANX, TRANY, TRANZ - translations
ROTX, ROTY, ROTZ - rotations
XSCALE#, YSCALE#, ZSCALE# - scaling
4) Other setup statements.
5) Setup screen for double buffering.
6) Animate object by changing control variables.
7) Call object calculation procedure.
8) Call object draw procedure.
9) Show new screen, and set work screen to the hidden screen.
Draw image:
1) Draw line from the starting point, LINEA(), to end point LINEB(). Use point data for the screen, stored in XP()

and YP().
Calculate image:
1) Get initial point definition.
2) Use initial point and scale about X, Y and Z.
3) Use scaled point and rotate about Z.
4) Use rotated point and rotate about Y.
5) Use rotated point and rotate about X.
6) Use rotated point and translate about X, then Y, and Z.
7) Use final translated point and use perspective to give screen X and Y co-ordinates.
8) Store screen X and Y co-ordinate in array, XP() and YP().

PROGRAM BREAKDOWN

Create screen 0 as low resolution in monochrome.
Switch cursor off and remove the mouse.
Define palette.
Define number of points and number of lines for 3D object.
Reserve space for 3D point data.
Reserve space for on-screen point data.
Reserve space for line information.
Set up global variables.
Initialize 3D object and variables.
Create double buffered screen.

Initialize transformation variables.
Initialize rotation variables.
Initialize scale variables.

Main animation loop.
Modify 3d object control variables.
Calculate new image.
Draw image.
Show new screen, clear old screen and then work on old screen.
Repeat until the mouse button is pressed.
End

Procedure to draw image using two parameters, these being the X and Y position for the image to be displayed at.
Repeat for all lines
Find start and end point of line.
Draw a line from start point to end point.
Note: each point is offset by X and Y which in this example are the centre of the screen.
End of procedure.

Procedure to calculate image using nine parameters, these being :-
ROTX, ROTY, ROTZ - angle of rotation for each axis
TRANX, TRANY, TRANZ - translation for each axis
XSCALE#, YSCALE#, ZSCALE# - scale factor for each axis.

Calculate sine and cosine for each angle ready for later on.
Repeat the following for all points in the image Scale point.
Rotate about Z axis.
Rotate about Y axis.
Rotate about X axis.
Translate point.
Calculate bottom half of perspective equation. If it equals 0 then it is set to 1. This will avoid a division by 0 error later on.
Calculate screen x and y co-ordinates according to perspective equation.
Store screen co-ordinates in array.
End of procedure.

Procedure to initialise.
Repeat for all points
Read and store 3D point data in array.
Repeat for all lines
Read and store point data in array.
Define centre of screen. This is only used to display image.
Define distance to centre of screen.
Set degree mode.

Data for points in 3D.
Data for lines.
End of procedure.

Feel free to change the program to suit your own needs. You might like to try different 3D objects or changing the main program for different animation effects.

Try adding any of the following to the main program (just after the line ADD ROTZ,4,0 to 359) :-
1) xscale#=xscale#+0.01
2) add tranx,4
3) add tranz,-4
NOTE: the current 3D routines only allow monochrome images and you cannot view the image from inside correctly.

To do the above will require the following extra routines :-
Clipping : allows you to go inside the image.
Depth sorts : allows multi-colour objects which I hope to cover in future articles.

```
While Screen<>-1
Screen Close Screen
Wend
Screen Open 0,320,256,2,Lowres
Curs Off
Flash Off
Hide On
Palette $0,$FFF
NUM_PTS=8
NUM_LINES=12
Dim X(NUM_PTS),Y(NUM_PTS), ➡
Z(NUM_PTS)
Dim XP(NUM_PTS),YP(NUM_PTS)
Dim LINEA(NUM_LINES),➡
LINEB(NUM_LINES)
Global NUM_PTS,NUM_LINES,➡
```

```
DIST#,XCEN,YCEN
GlobalX(),Y(),Z(),XP(),YP(),➡
LINEA(),LINEB()
_INITIALIZE
Cls 0
Double Buffer
Autoback 0
TRANX=0 : TRANY=0 : TRANZ=0
ROTZ=0 : ROTY=0 : ROTZ=0
XSCALE#=1
YSCALE#=1
ZSCALE#=1
Repeat
Add ROTZ,4,0 To 359
Add ROTY,4,0 To 359
Add ROTZ,4,0 To 359
_CALC_IMAGE[ROTZ,ROTY,➡
ROTZ,TRANX,TRANY,TRANZ,➡
XSCALE#,YSCALE#,ZSCALE#]
_DRAW_IMAGE[XCEN,YCEN]
Screen Swap
Wait Vbl
Cls 0
Until Mouse Key
End
Procedure _DRAW_IMAGE[X,Y]
For LINE=1 To NUM_LINES
PNT1=LINEA(LINE)
PNT2=LINEB(LINE)
Draw XP(PNT1)+X,YP(PNT1)+Y ➡
To XP(PNT2)+X,YP(PNT2)+Y
Next LINE
End Proc
Procedure _CALC_IMAGE[ROTX,ß
ROTY,ROTZ,TRANX,TRANY,➡
TRANZ,XSCALE#,YSCALE#,➡
ZSCALE#]
CX#=Cos(ROTX) : SX#=Sin(ROTX)
CY#=Cos(ROTY) : SY#=Sin(ROTY)
CZ#=Cos(ROTZ) : SZ#=Sin(ROTZ)
For PNT=1 To NUM_PTS
X=X(PNT)*XSCALE#
Y=Y(PNT)*YSCALE#
```

```
Z=Z(PNT)*ZSCALE#
X1#=X*CZ#-Y*SZ#
Y1#=X*SZ#+Y*CZ#
Z1#=Z
X2#=X1#*CY#-Z1#*SY#
Y2#=Y1#
Z2#=X1#*SY#+Z1#*CY#
X3#=X2#
Y3#=Y2#*CX#+Z2#*SX#
Z3#=Z2#*CX#-Y2#*SX#
X4#=X3#+TRANX
Y4#=Y3#-TRANY
Z4#=Z3#-TRANZ
DD#=Z4#+DIST#
If DD#=0 Then DD#=1
X#=X4#*(DIST#/DD#)
Y#=Y4#*(DIST#/DD#)
XP(PNT)=X#
YP(PNT)=Y#
Next PNT
End Proc
Procedure _INITIALIZE
For I=1 To NUM_PTS
Read X(I),Y(I),Z(I)
Next I
For I=1 To NUM_LINES
Read LINEA(I),LINEB(I)
Next I
XCEN=160
YCEN=100
DIST#=250
Degree
Data -20,-20,-20
Data 20,-20,-20
Data 20,20,-20
Data -20,20,-20
Data -20,-20,20
Data 20,-20,20
Data 20,20,20
Data -20,20,20
Data 1,2,2,3
Data 3,4,4,1
Data 5,6,6,7
Data 7,8,8,5
Data 1,5,4,8
Data 3,7,2,6
End Proc
```

RUNNING RUMOURS

Remember how I told you about the new AMOS music extension which allows you to play Soundtracker files in the last issue? Well, I have some more information!

It is specifically aimed at playing Startrekker modules, plus chip sound effects created with the same package!! Anyway, until it is freely available you can get a copy of the extension if you have bought Fun School 4!!

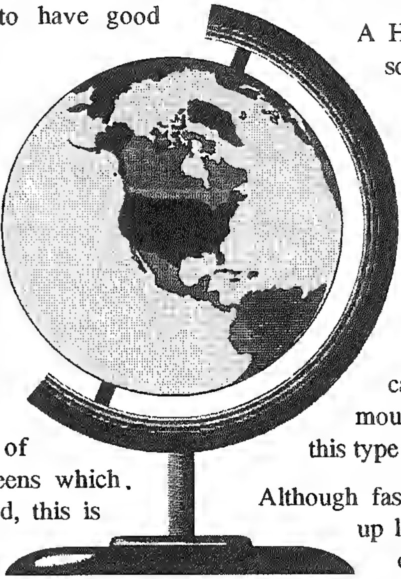
On two of the Fun School 4 packages (Under 5s and 5-7s) the programs have been stored as uncompiled AMOS files, this means you can load them into AMOS and have a play with them! The new music extension is stored inside the "AMOS_SYSTEM" folder and is called "MUSIC.LIB". It is a direct replacement for the current music extension, so all you do is copy it into your own "AMOS_SYSTEM" folder. You can find out how to use it by looking through the FS4 programs. Hopefully I should have a list of commands and more info on this ready for the next issue.

WINDOW ON THE WORLD

BY KYLE REES

Many people bought AMOS with the belief that it would help them create fantastic software with ease, save the universe as we know it and brew the perfect cup of coffee. Unfortunately life is never that simple and to get any satisfaction from using this language you will have to get to know your Amiga and its capabilities a little better. It also helps to have good Colombian beans.

Some of the most amazing games available for our not-so-humble hardware hotpot exhibit smooth fast scrolling. I used to rather enjoy a game of Xenon II, but my brother is a great fan of the Kick Off football game. Both of these games have screens which, constantly move around, this is what we mean by a SCROLL.



There are two main methods of producing a scrolling screen. The first is to take a series of interconnecting picture "blocks", place one on the screen, move it a bit and then place the next section. This is known as a SOFTWARE SCROLL, because it is up to us (the programmers) to do all of the hard work. Xenon II uses this type of scrolling.

One of the advantages of software scrolling is that it requires very little memory because you really only have to have a standard sized screen open which is 320*200 pixels in size. A 16 colour screen at this resolution would use up 32k and when we double buffer it to make our BOBs smooth and flicker free it would use up 64k.

A HARDWARE SCROLL is something which is handled mainly by the Amiga's custom chips (which are known as hardware). It is produced by defining a big picture, looking at one section of the picture and then panning across the rest, just like a television camera looking at a wall mounted mural. Kick Off uses this type of scrolling.

Although fast, hardware scrolling eats up lots of memory. If we had created a big screen with about 16 colours and covering an area of 640*200 pixels it would take up 64k and when we introduce double buffering it increases to a massive 128k!!

A SCREEN WITH A VIEW

Fortunately the ease at which we can manipulate a hardware scroll more than makes up for the memory consumption. Before we do this I think we should look at how versatile the Amiga screen system really is.

Amiga screens are something special, you need to imagine them not as literal pictures but as a window looking into your machines memory. By moving our viewpoint around we can examine almost any part of the chip memory contained inside the Amiga. Incidentally, this is how many of the graphics "grabbers" work.

AMOS allows us to manipulate these "windows on the world" using the SCREEN DISPLAY and SCREEN OFFSET commands. SCREEN OFFSET controls our view of the computers memory through our "window", while SCREEN DISPLAY controls the size of this "window" and its position on the monitor.

Ok, now we all know how to open a screen inside AMOS don't we? It's pretty simple, load AMOS and make sure you are inside the editor (not DIRECT MODE). Type this line in:

```
Screen Open 0,320,200,16,Lowres
```

This tells AMOS to create screen number 0 with a horizontal resolution of 320 pixels, a vertical resolution of 200 pixels, 16 colours in LOW RESOLUTION mode (which means the pixels are pretty square and chunky). We now have a screen we can experiment with.

The next step is to define two variables XSIZE and YSIZE, these will tell AMOS how big our "window" will be.

```
XSIZE=320
YSIZE=200
```

Ok, now we must start a loop which will allow us to change the characteristics of this "window" with the mouse buttons. Every time we go around the loop the mouse coordinates are put into the variables X and Y. These are then used to position the "window" on your monitor. It sounds confusing, but if you imagine the "window" as a piece of paper being moved around on a desk (which is your monitor!) it may seem a little clearer.

```
Repeat
X=X Mouse-100
Y=Y Mouse-160
```

Now we must check the state of the mouse keys. If the left button is pressed the variable XSIZE is decremented, if the right button is pressed the variable YSIZE is decremented. The command ADD allows a little more control over variables than the standard INC or DEC commands. By using ADD we can make the number contained in XSIZE wrap around from 0 to 320 with very little work.

```
If Mouse Key=1
Add XSIZE,-1,1 To 320
End If
If Mouse Key=2
Add YSIZE,-1,1 To 200
End If
```

Ok, now for the magic command. As you can see the first parameter is the screen number, followed by the X & Y positions plus XSIZE and YSIZE.

```
Screen Display 0,X,Y,XSIZE,YSIZE
```

The final part of our program waits for you to press both mouse keys before it stops.

```
Until Mouse Key=3
End
```

I hope this has not been too complicated to follow. The concepts are simple, but I must admit that the whole subject can become quite overpowering!

I have now introduced you to the basics of using SCREEN DISPLAY, but what about SCREEN OFFSET? It is this command which actually creates the scrolling. If we defined a standard sized screen and used SCREEN OFFSET to look above it (or below it) we would see what is contained in those portions of memory.

To start off with, let's open up our screen by typing the following lines into the AMOS editor (remember to save your last program if you want to keep it).

```
Screen Open 0,320,600,4,Lowres
Flash Off
Hide On
Curs Off
Palette $0,$90
Cls 1
```

You will notice that I have reduced the amount of colours from the standard 16 down to 4. I have also increased the size of the screen to 600 pixels high (which is 3 times the height of a normal screen).

The next SCREEN DISPLAY command features four commas in succession. This is NOT a printing mistake! It tells AMOS to use it's own defaults instead of the missing parameters. The last part of the line tells AMOS that we only want to display 200 lines of this 600 line screen.

```
Screen Display 0,,,,200
```

I am not a great lover of football (or football simulation computer games) but they can be useful to demonstrate the principles we are dealing with here. So in the best tradition of Blue Peter, here is a pitch I drew earlier....

```
Draw 0,299 To 320,299
Box 0,0 To 319,599
Circle 160,299,105
```

Even though we cannot see the whole screen, AMOS will draw it for us. Neat huh? The penultimate step in this program is to let AMOS know which part of the 600 line screen we wish to display inside our small 200 line "window". Once again the SCREEN OFFSET command contains a couple of commas, this tells AMOS to use the default value for the X position of our view.

```
YPOS=0
Screen Offset 0,,YPOS
```

Now for the exciting bit!! By pressing the left mouse key the screen will scroll smoothly upwards by telling AMOS (through the SCREEN OFFSET command) to look at a different part of the picture held in memory through our small screen or "window". If we press the right button the screen will scroll downwards.

```
Repeat
If Mouse Key=1 and YPOS>0
Add YPOS,-5
Screen Offset 0,,YPOS
End If
If Mouse Key=2 and YPOS<400
Add YPOS,5
Screen Offset 0,,YPOS
End If
```

We must put a WAIT VBL command here to tell AMOS to slow down. Try removing it to see how fast this program runs!!

```
Wait Vbl
Until Mousekey=3
```

In the next issue we will take a brief look at horizontal hardware scrolling. If you have any questions about hardware scrolling, please write in and we will try to answer them in two months time.

HANDY HINTS

Ok, so nobody is perfect. The original AMOS manual does have a few errors(!!) most of them due to the fact that AMOS was still being written almost up to its day of release!

One of the worst boo-boos is the inclusion of two commands which don't even exist!! These are:

WINDOW FONT

and

LLIST

So if you've been wondering why they just don't work, now you know!

ATARI ATTACK!!

There is a vast amount of freely available graphic collections for other 16 bit machines, especially the Atari ST. With something like CrossDos or MessyDos you can read these disks, but you cannot display the pictures. Well now you can, with this amazing program written by Terry Mancey. It will read in and display any picture saved in Neochrome (a popular ST painting package) format.

```
Screen Open 0,320,200,16,Lowres
Flash Off
Curs Off
Reserve As Work 15,32128
F$=Fsel$("*.NEO","","")
Bload F$,15
_SHOW_NEO[15]
Wait Key
Default
End
Procedure _SHOW_NEO[BANK]
PALT=Start(BANK)+4
For C=0 To 15
Colour(C),Deek(PALT+(C*2))*2
Next C
PICT=Start(BANK)+128
For Y=0 To 199
For X=0 To 19
Doke Phybase(0)+(X*2)+(Y*40), ➡
Deek(PICT+0)
Doke Phybase(1)+(X*2)+(Y*40), ➡
Deek(PICT+2)
Doke Phybase(2)+(X*2)+(Y*40), ➡
Deek(PICT+4)
Doke Phybase(3)+(X*2)+(Y*40), ➡
Deek(PICT+6)
Add PICT,8
Next X
Next Y
End Proc
```

AMOS ASSEMBLER

The continuing story of Gary Symons' fight against ignorance, injustice and decaffeinated coffee

Before I start, I would like to take time out to answer a question. I was asked why the macro power program supplied with the AMOS Assembler in the licenseware does not work in AMOS 1.3. This is because the internals of AMOS 1.2 which macro power does work on are completely different from AMOS 1.3 internals. Francois made these welcome changes for the compiler. They are welcome to a programmer (such as myself) who writes within AMOS as it is now very modular (library wise). Every time we used to change AMOS we had to assemble the whole lot taking 3 minutes on a standard A500, but now we can work on specific sections.

Last time we took a look at binary. Binary is a base 2 number system which is used by assemblers mainly because we can see which Bits are on and off. For example there is a memory location in the AMIGA which controls which disk drive motors are turned on. A Bit which is on (1) in a certain position tells me whether the drives currently selected are on or off. Numbers aren't generally looked at in binary because large numbers are 16 to 32 Bits long and that takes up a lot of room on the screen so we use decimal. However when we wish to use a number and know its Bit status we use a system which is short to write but easy to convert to binary mentally. If you try and convert decimal to binary you must divide the number by two, noting the remainder each time, and forming a binary number from right to left. This is time consuming when you are writing a program. The system we use is called hexadecimal (base 16) numbers 10 to 15 are denoted by A to F, so 14 is E in hex (we write $E the $ signifying hex). So what is 16 decimal in hex? Well its $10, so 31 is 16+15=$10+$F=$1F. Our decimal system base 10 carries over into the next left digit when we go over 9; in base 16 we carry over after 15 $F. The reason hex is used is that we can see from any hex number the Bits set. $FF is 255 in decimal, the hex version tells me (since each hex digit is 4 Bits long, you work it out!) that all 8 Bits are set. It's interesting to realise that computers always work with Bits and humans can interpret these Bits in many different ways. There is another number system where 3 Bits represent one digit called octal (base 8). It's slightly longer than hex and is just as useful for smaller numbers.

Binary numbers have interesting properties. If we shift the binary number one place to the left and add a 0 Bit to the end we are in effect multiplying that number by 2. If we shift it right removing the right most Bit we are in effect dividing by two. The 68000 has these operations as part of its machine code instruction set, and since this is done by the hardware of the 68000 chip it is very fast. You may be relieved to know that general purpose multiply and divide instructions exist also on the 68000 chip, whereas in 6502 we used to have to write our own multiply and divide instruction (when we write our own instruction we usually call it a subroutine or sometimes a macro. There is a significant difference. A macro name may represent many assembler instructions and each time that macro name is found by an assembler it will substitute the instructions and assemble them at that point. A subroutine is a small program which can be called from anywhere in your main program and when finished will return to the main program) using multiple adds and subtracts and shifts; and the C.P.U. then could only handle 8 Bit numbers meaning larger numbers had to be split up into 8 Bit sections. The latest generation of chips are handling strings and floating point numbers and make light work of 32 Bit numbers.

When working in assembler you will come across logical operations. Logical operations AND, OR, and EOR (Exclusive Or) operate on the Bit as follows.

Take two Bits A and B these can be 1 or 0 the result C can also be 1 or 0.

C = A AND B The result C is 1 If A and B are both 1 so 0 AND 0=0, 0 AND 1=0, 1 AND 0=0, 1 AND 1=1

C = A OR B The result is 1 If A OR B is 1

C= A EOR B The result is 1 If A is 1 Or B is 1 BUT not both so 1 EOR 1=0.

A, B, and C can be more than one Bit long. In this case we perform the operation Bit by Bit.

For example

A=1101 B=1001 C = A EOR B

A 1 1 0 1 EOR B 1 0 0 1 - C 0 1 0 0

We operate on each column in turn.

Logical operations are the foundation of the machine code operations, in fact designers of machine code and custom chips such as the Blitter (a custom chip in the AMIGA which can move chunks of memory around at high speed shifting all the Bits at the same time if required) use this logic creating the chips operations. Circuits are designed and tested on computer programs before they physically exist. The computer programs emulate the hardware using these logical operations which are available as part of the machine code instruction set being used by the program itself. When the designer is happy with the tests, the computer can produce the chip using automated external devices. The computer programs can also optimise circuit design making the chips smaller and use less power. Efficient design can save hours on the battery life of a hand held computer.

When the supercomputers with parallel processing appear we will be able to program our own custom chips such as Blitters, allocating a parallel computer for that task. This means that all previous computers such as the AMIGA could be emulated (legality aside) with no problems in running at a rate 100 times faster (or more).

In the last issue we saw how memory locations are used to store data. The 68000 C.P.U. (Central Processing Unit) has to 'communicate' with memory and can get 'held up' while other chips such as the Blitter are using memory. Thankfully a C.P.U. usually has its own storage area. Each location in this area is called a register. The 68000 has 16 32 Bit registers, each register can perform operations such as the ones described above with other registers. The C.P.U. can perform operations on memory locations and can also operate on memory with registers and vice versa but as I stated above although very fast by high level standards, register operations are much faster than memory ones. Moving data from memory to a register, or vice versa, is probably the most used of all the machine operations. Jumping to other areas of the program or branching is also very common as is returning from a subroutine, which brings me conveniently to the stack ("What a link!"- Ed.).

The stack is an area of memory pointed to by one of the registers called a stack pointer which shouldn't usually be changed by you, but all the others are free to use. The stack is useful for keeping temporary numbers and returning addresses telling the C.P.U. what to change the program counter to once the subroutine is finished. When you call a subroutine from your main program the next address after the subroutine branch instruction is stacked so that once the subroutine is finished the returning address can be found. As I have explained each register is 32 Bits long but we don't always have to use the operations on all 32 Bits. We can restrict them to BYTE (the first eight Bits), WORD (the first 16 Bits) or LONG (all 32 Bits).

MY DEFINITION

BITPLANE

Screen memory on the AMIGA is made up of Bitplanes. A Bitplane is a chunk of memory which, when the video hardware is pointed to, decodes into pixels. A one Bitplane screen shows up as a 2 colour picture on the monitor. Each memory location has 8 Bits which are decoded into 8 pixels. If a Bit is 1 then colour 1 is used, if 0 then colour 0 (the background colour) is used. If we use more Bitplanes we get more colours because multiple Bitplanes are looked at in 'parallel'. Two Bitplanes result in 2 Bits being looked at, 1 of the 2 Bits from each 1 of the 2 Bitplanes resulting in a 2 Bit number for each pixel which can have up to 4 values thus a four colour screen.

Let's use a 1 Bitplane screen to explain how pixels are organised. The first BYTE of memory locations in the Bitplane is the first top left 8 pixels on the screen so if we poke 10000000 in that first BYTE we will get the first pixel illuminated, 00100000 the third pixel the second BYTE represents pixels 9 to 16 and then next row is at BYTE 40 on lowres (BYTE 0 is the first BYTE). In hires it's BYTE 80 on the next line. Lowres 40*8=320 pixels across, Hires 80*8=640 pixels across.

I will now show you a program which will give a shifting illusion in a one Bitplane screen.

```
Set Buffer 50
Procedure _ASSEMBLER[A$,    ➡
_ASM_VAR1,_ASM_VAR2,        ➡
_ASM_VAR3]
```

(The next procedure "_COMPILED", is only to be included if you have the version of the AMOS Assembler which comes with the AMOS Compiler)

```
Procedure _COMPILED
Reserve As Work 14,$1000
_ASSEMBLER["initialise label ➡
buffer",$1000,0,0]
Screen Open 0,640,200,2,Hires
Colour 1,$FFF
Curs Off
X$=""
X$=X$+"Set_Screen:"
X$=X$+"move.l a0,a1;"
X$=X$+"move.l #80*200-1,d1;"
X$=X$+"loop:move.b d0,(a0)+;"
X$=X$+"dbf d1,loop;"
X$=X$+"rol.b #1,d0;"
X$=X$+"move.l a1,a0;"
X$=X$+"rts;"
_ASSEMBLER[X$,Start(14), ➡
Start(14)+Length(14),True]
Print
Print
Centre "Left mouse key for ➡
high res or right for low!"
Repeat
K=Mouse Key
Until K=1 or K=2
If K=1
M$=" high res "
Screen Open 0,640,200,16,Hires
Else M$=" low res "
Screen Open 0,640,200,16,Lowres
End If
Curs Off
Colour 1,$FFF
Paper 0
Pen 1
Areg(0)=Phybase(0)
Dreg(0)=1
Timer=0
For I=1 To 100
Call Start(14)
Next I
Print "Took";Timer; ➡
"vertical blanks in";M$;"mode!"
```

Make sure you have the assembler procedures, the interpreted version has one procedure only.

The assembler part is in a string.

```
X$=""
X$=X$+"Set_Screen:"
X$=X$+"move.l a0,a1;"
X$=X$+"move.l #80*200-1,d1;"
X$=X$+"loop:move.b d0,(a0)+;"
X$=X$+"dbf d1,loop;"
X$=X$+"rol.b #1,d0;"
X$=X$+"move.l a1,a0;"
X$=X$+"rts;"
```

Notice that the basic program set AREG(0) which is address register a0 to the address of the first Bitplane of the screen. There are 8 address registers a0 to a7 and 8 data registers d0 to d7. The address registers are used to reference memory whereas the data registers are used to store and hold data and to have operations such as logical or mathematical ones performed on them.

The first line of the assembler program held in the string moves the contents of a0 to a1 thus making a copy in a1. The .l is the size of the transfer meaning LONG (i.e. all 32 Bits).

The data register d1 has the number of BYTEs -1 in the Bitplane this is because the first screen BYTE can be thought of as BYTE 0.

At the label loop the BYTE part (.b) of d0 is moved into the memory location pointed to by a0 the brackets around a0 means location a0. The + means that after the transfer a0 is incremented by 1, as only a BYTE has been used. Had it been a WORD then 2 would have been added and LONG WORD would have meant an addition of 4.

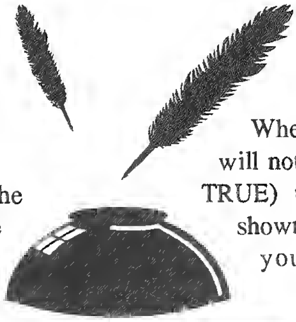
The line with dbf d1,loop means decrement d1 and if not -1 then branch to the address pointed at by the label loop.

The line rol.b #1,d0 means rotate the BYTE in d0 (the first 8 Bits) once to the left. If the Bit at Bit 7 in the BYTE is rotated out it is put back into Bit 0 (8 Bits, Bits 0 to 7).

The line move.l a1,a0 restores the value of a0 to the beginning of the Bitplane for the next time the routine is entered.

Finally rts will return AMOS back to the line after the call.

Briefly, a Bit moving across a BYTE gives the illusion of movement across 8 pixels. Since all the BYTEs in the screen are doing this there is an illusion of the screen scrolling left.



If you have an assembler related question why not drop Gary a line? Unfortunately he will only be able to answer through this column.

When you run the program you will notice (since I have set listing to TRUE) that the assembler listing is shown. To the left of the listing you will see hexadecimal numbers. The first number is a memory location and the hexadecimal WORDs (16 Bit numbers) after the address are the contents starting at that address. If more than one WORD is shown on a line, just add 2 to the address mentally for each WORD after the first. This will give you the address that contains that WORD. The next line starts with the address+2, if only one WORD was shown on the line above; if 2 were shown then it would be +4. Those WORDs are the machine code which has resulted from the assembler changing the mnemonics held in the string X$ to machine code. If you look at an assembler manual or handbook, the way these numbers are formed are explained. The mnemonics are a standard for 68000 used by all 68000 assembler programmers, so any 68000 manual will use the same mnemonics. Different processors such as the 8086 on the PC computers use different mnemonics and the resulting machine code is different from 68000, however 68000 assembler can be converted to other processors by what is known as cross assembly. A cross assembler changing 68000 assembler to 8086 for instance will mean that each 68000 mnemonic is converted into one or more 8086 machine code instructions. Cross assembled code used to be slower than original assembled code but sometimes it can be as fast or faster. It's sometimes faster because old programs running on old, slow machines are converted to faster machines. For instance conversion of 68000 assembler to the pipelined (executing more than one function at a time) RISC system on the Archimedes results in a much faster program.

There is a major problem with conversion and that is that different computers have different hardware. Hardware is usually accessed by specific memory locations, the contents of which operate the hardware. These locations differ from machine to machine, not taking into account the fact that machines usually have different types of customised hardware specific to each machine. This is one of the reasons why C was created. A standard C library which uses the hardware usually comes with C, so a transfer of C source code from one machine to another is all that is needed, but C isn't as fast as some people make out! Anyway, back to the program.

Study the program and get yourself an assembler manual describing all the 68000 instructions. You could use other operations instead of rotate to get different results.

By using a bigger sized move the speed can be improved. The size of d1 would have to be halved for WORD moves and divided by 4 for LONG WORD moves. If you move WORDs or LONG WORDs into odd numbered memory locations (Bit 0 set in address) the 68000 will generate an error resulting in your machine giving a guru sign (denoted by the number 3 followed by the address where it occurred).

The program also shows the speed differences between high res and low res for the same operation. Can you work out why?

MY DEFINITION

<u>AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE (ASCII)</u>

This complex acronym is the name given to the internationally accepted system of storing letters of the alphabet, numbers, punctuation and other characters on computer using binary codes. The characters between 32 (binary number 00010000) to 127 (binary number 00111111) are standard numbers and letters of the alphabet, with the characters from 0 (00000000) to 31 (00001111), and 128 (01000000) to 255 (01111111) being reserved for special characters and control codes which are used by the computer.

SOFTWARE DEVELOPMENT (THE EUROPRESS WAY)

BY RICHARD VANNER

In this article I want to let you know how a project starts off from an idea and progresses through development into a fully commercial package. I'd like to sing the praises of all the un-sung heroes at Europress without whom a project would never be completed.

THE IDEA

Most ideas evolve over time. It's not a sudden desire to start up a project when the light bulb appears above someones head. AMOS 3D is a good example of a project that evolved. When STOS and AMOS had obviously gone down so well, we knew a 3D extension would be the ideal subject that users would want to tag onto their language. So as this idea was banded around and people had time to visualise the project in their minds, action was taken to recruit a team that could deliver the goods.

Most of the idea stage is created in the development department. The higher management team would then consider the possibility of such a project and also add to its specifications. We work as a team, discussing the pros and cons of a project - will the consumer really want this product? Will it take up valuable internal time to get produced? Many things are considered before a project hits the road towards release.

DEVELOPING

All projects are developed along a standard path. A team is assembled for writing the package, hardware acquired for the team to work on and a project manager takes responsibility of the program. Once under way it is a juggling act to keep things going as fast as possible. Peter will tell you that we like our programmers to be flexible (by working 25-hour days, for example). But at the end of the day it is essential to the success of the product that the team is fully motivated and enjoy writing the product.

A spec in most cases, is completed fully before the project begins. This will change and evolve through the course of the development cycle. New ideas are added, bad ones discarded and most importantly, if the opposition has a good feature, we'd better add it to ours!

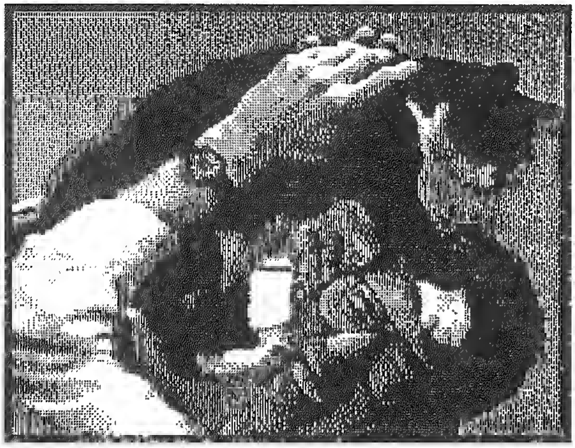
Manuals are also a major task for us. Most programmers, although good at explaining in detail what their program does, cannot add the natural feel and narrative style of a professional writer. A great example of this is Easy AMOS. François, even though he speaks adequate English to communicate his ideas and to document his programs, is obviously no use when in comes to preparing a first draft of the manual. In the case of Easy AMOS it is even more important to have a style which is humorous and easy to read. We therefore hunted down Mel Croucher, a well known character in the industry who has the perfect style for such a manual. Luckily for us he was just coming free and could take on the job. In fact this is the key to good project management: using freelancers gives us extreme flexibility of our resources.

Okay, so the software is started, at an appropriate point the manual is contracted and we're off! But now we enter the tricky stage. This is when everyone in the company knows that the project exists for real and is targeted for release in the near future. People start asking questions, ones that have no answers until some of the future arrives. Everyone has their own ideas about what the project is and they pull the project manager towards their vision - it is now up to the project manager's discretion to decide what's in and what's out.

I could write a whole book on this subject and I think I will one day, but for now let's assume all goes well, the programmers stay happy, the manual author escapes death and the project keeps on budget (ahhhmmm).

MARKETING

Paul Shrimpling is Europress' Marketing Manager. It is his responsibility to advertise any future and current projects in the various magazines. Paul along with his (biscuit-nicking) assistant Anna Donaldson, also have to work on box designs, press releases, liaising with IMPACT (a team of merchandisers), creating Fun School stickers, the teas and coffees,



you name it they do it!

So the look of a package comes from this department. A project manager receives the designs which are then copied from, to create the screen graphics.

PRODUCTION

So now let's imagine our product is almost complete. The Project Manager must now talk to Richard Peacock, the Production Manager, about the product's components. It is Richard's job to ensure that X thousand discs labels, manuals and boxes will be ready on the release day.

Using external duplicators and printers, the product's items are made ready for release. When all items are complete they all end up at a company called Multi-pac, here the items are boxed together and then delivered to Europress ready for shipping out to distributors and dealers.

SALES

Headed by Diane O'Brien, her team aim for maximum sales of the product in all corners of the globe. They all interact with development to see what the product is about and learn its key features. They prepare the way for the product's success.

Whereas development is 100% male, this department is 100% female. Their team includes: Clare Barnwell, Nicola Murray and Amanda Blackwell.

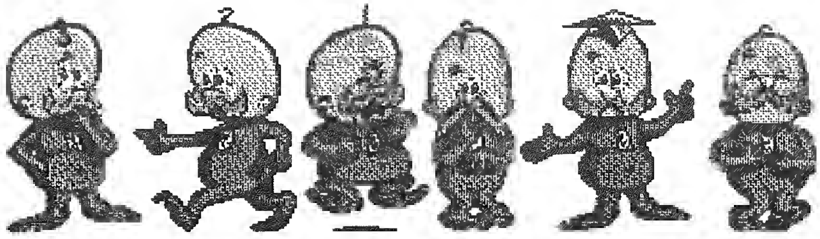
DESPATCH

Once Richard Peacock has got the stock in and Sales have raised the orders, it's now time for the team in despatch to fulfil their side of the project by sending out the goods. For all the thousands of units sold this team comprises of only three members: The always jovial Barbara Blackshaw, David (Stockport County) Middlemas and Gary (Kick-Off mad) Russell. In December, our busiest month, the three of them manage to send out more than 50,000 products.

CUSTOMER SUPPORT

A vital part of our organisation and one which we try to improve day-by-day. Answering the phones, letters and Software Returns are Barbara, Dot, Alison, Mad Pam and Julie. If the question gets a bit tricky they will then interact with development, calling up the help of Lee 1, Lee 2, Alex or even the project managers.

With products as complex as AMOS and Mini Office we certainly end up with a lot of mail every week. But due to our experience in this field we turn enquiries around within a maximum of two days.



A SNEAK PREVIEW OF THE EASY AMOS CHARACTER, WEIRD HUH?

OTHER DEVELOPMENT DETAILS

Many projects are happening within development. I work with Alex who, at 18, has almost completed his first major project: STOS 3D.

Marc Dawson is the Senior Project Manager and ensures we're all doing what's important and not just evaluating AMOS games! He also has Arron 'snowboard' Maclean to help him out. Well done Arron - no one else wanted to drive for three hours to Corby with the Fun School 4 masters!

Dave Thomas is a Projects Coordinator and he works with Lee Fahy on various programs. Lee has had two years in Customer Services and so he knows what our customers want and like.

We also have two amazingly talented programmers in-house. Gary Hughes and Darren Ithell are responsible for creating program ideas, which can be easily tailored and controlled. Once the designs are finished we then simply need to get them converted to other formats.

PAY DAY

Well we all work for it and so we need a department to deal with it. The accounts department is headed by Ian McFegan (who is one of our guinea pigs on Easy AMOS - to ensure that complete beginners can create programs with ease). John (I'm gunna score on Saturday) Turnbull assists him as well as supplying mounds and mounds of dosh for expenses. But the most patient of the three must be Sue. She has to deal with Invoices (sometimes on toilet roll), chase our customers for £2.5 million in bills a year, and general paperwork that comes out of development - no wonder she only works half days!

I nearly forgot our longest serving piece of furniture! A classy lady who drives a BMW. Christine Aspin enters the orders from sales onto our mega accounts computer and also feeds us Aspirins when we have headaches from all the eye strain of computer glare!

THE BOSS

The most tuned-in and smiling guy is Chris Payne. As M.D. of the company he runs the whole show. But not without great consideration for what his employees have to say. Without an M.D. who understands what Software is all about you won't get far (Ok, I think you've got your raise now Richard! Ed.).

HANDY HINTS

Sometimes AMOS is just too fast for its own good, especially if you are using Dual Playfield mode!!! If you have been having problems getting it to work properly, with strange corrupted screens appearing, remember to put a WAIT VBL command directly before the DUAL PLAYFIELD command.

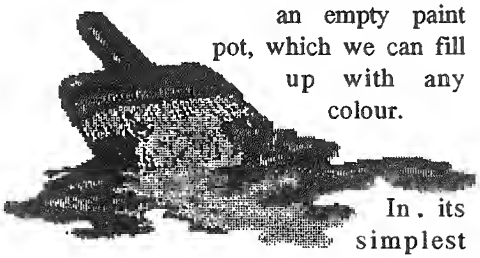
You may encounter similar problems if you are using AMAL and testing it using the CHANAM(), CHANMV() or MOVON() functions. These will not return the correct values unless you do a WAIT VBL straight after the AMAL ON command.

SPLISH, SPLOSH

BY PETER HICKMAN

Your Amiga is blessed with a wonderful collection of 4096 different colours. With these colours we can create new worlds and alternate realities with art packages like Deluxe Paint. With AMOS we can go a step further and use the power of colour to produce animation.

Before we go into too much detail it will help for you to know a little bit about the way colours are handled by the Amiga. Under ordinary circumstances you can display 32 individual colours on a screen, these are stored in colour REGISTERS. You can think of a colour register as being an empty paint pot, which we can fill up with any colour.



In. its simplest form you can see colour cycling at fairgrounds and fancy Christmas displays, this happens when a string of unlit bulbs are switched on in sequence one at a time. This gives the illusion that a bulb is moving along. We can demonstrate this with a simple program.

The first step is to open up our screen, so go into the AMOS editor and type this

```
Screen Open 0,320,200,16,Lowres
Flash Off
Cls 0
Curs Off
```

Now we must set up our palette. Remember that each of the numbers in the following line represent that "paint" that we are pouring into the paint pot (colour register). The colour which we have poured into colour register number one (remember that the 32 different colour registers are labelled from 0 to 31) is white ($FFF) and represents our bulb which is switched on.

```
Palette $0,$FFF,$0,$0,$0,$0,$0, ➡
$0,$0,$0,$0,$0,$0,$0,$0,$0
```

Now we must draw our lightbulbs, each one will use a different colour (selected with the PEN command).

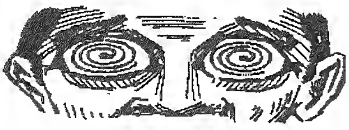
```
Paper 0
For LOP=1 To 15
Pen LOP
Wait Vbl
Print "O ";
Next LOP
```

The final stage is to start the colour cycling up. We will do this using the SHIFT command. This command uses four parameters, the first is the speed at which the colours will cycle, the second and third specify the start and end colour registers which will be cycled and the final parameter for special effects (which we will not be using!).

Just how does it work? Well, the SHIFT command will move the colour stored in register number 1 into register number 2, the colour in register 2 into register 3, 3 into 4, 4 into 5 etc. It's a bit like pouring the paint from one pot into another!

```
Shift Up 2,1,15,1
Direct
```

That's it. Simple, huh?



Another interesting use of colour cycling is to produce weird backgrounds for games and demos (or hypnotising friends and relatives!!).

In the next example we will draw a small block 16*16 pixels in size, grab it as a bob and finally paste it all over the screen. Once again our screen must be opened up, and a palette selected. In this case I have used shades of blue.

```
Screen Open 0,320,200,16,Lowres
Flash Off
Cls 0
Palette $0,$5,$6,$8,$9,$B,$C,$E,$F
```

Now we will read in the data (sorry, but there is quite a lot of it!) and draw it onto the screen.

```
For Y=0 To 15
For X=0 To 15
Read SPOT
Ink SPOT
Plot X,Y
Next X
Next Y
```

By using the GET BOB command we can grab this small block and stick it all over the screen, 20 blocks across and 12 blocks down.

```
Get Bob 1,0,0 To 16,16
For Y=0 To 11
For X=0 To 19
Paste Bob X*16,Y*16,1
Next X
Next Y
```

We must now start that funky colour cycling!! This time we will be cycling colour 1-8.

```
Shift Up 4,1,8,1
End
Data $1,$1,$1,$1,$1,$1,$1,$1,$1, ➡
$2,$3,$4,$5,$6,$7,$8
Data $1,$2,$2,$2,$2,$2,$2,$2,$1, ➡
$2,$3,$4,$5,$6,$7,$7
Data $1,$2,$3,$3,$3,$3,$3,$3,$1, ➡
$2,$3,$4,$5,$6,$6,$6
Data $1,$2,$3,$4,$4,$4,$4,$4,$1, ➡
$2,$3,$4,$5,$5,$5,$5
Data $1,$2,$3,$4,$5,$5,$5,$5,$1, ➡
$2,$3,$4,$4,$4,$4,$4
Data $1,$2,$3,$4,$5,$6,$6,$6,$1, ➡
$2,$3,$3,$3,$3,$3,$3
Data $1,$2,$3,$4,$5,$6,$7,$7,$1, ➡
$2,$2,$2,$2,$2,$2,$2
Data $1,$2,$3,$4,$5,$6,$7,$8,$1, ➡
$1,$1,$1,$1,$1,$1,$1
Data $1,$1,$1,$1,$1,$1,$1,$1,$8, ➡
$7,$6,$5,$4,$3,$2,$1
Data $2,$2,$2,$2,$2,$2,$2,$1,$7, ➡
$7,$6,$5,$4,$3,$2,$1
Data $3,$3,$3,$3,$3,$3,$2,$1,$6, ➡
$6,$6,$5,$4,$3,$2,$1
Data $4,$4,$4,$4,$4,$3,$2,$1,$5, ➡
$5,$5,$5,$4,$3,$2,$1
Data $5,$5,$5,$5,$4,$3,$2,$1,$4, ➡
$4,$4,$4,$4,$3,$2,$1
```

```
Data $6,$6,$6,$5,$4,$3,$2,$1,$3, ➡
$3,$3,$3,$3,$3,$2,$1
Data $7,$7,$6,$5,$4,$3,$2,$1,$2, ➡
$2,$2,$2,$2,$2,$2,$1
Data $8,$7,$6,$5,$4,$3,$2,$1,$1, ➡
$1,$1,$1,$1,$1,$1,$1
```

Have fun with colour cycling and send in any examples that you come up with. We will publish the best ones for everyone else to enjoy!!

In the next issue I will be taking a look at animation using screen swapping, a technique which goes back to the grass roots of smooth cartoon movement. It can take up quite a bit of memory so have those RAM upgrades ready!!

HANDY HINTS

When you are typing listings in from "ALL ABOUT AMOS" remember that the ➡ symbol is there to let you know a program line continues on the next page or the next line down.

The reason for this is simple- SPACE!! To keep the amount of white space to a minimum we are forced to use thin columns (like this one) for programs, otherwise most of the magazine would be blank.

If you have trouble typing in any of the listings, please write in and let us know what your problem is. We will try to sort it out as quickly as humanly possible.

FADING FAST!

Ok, it's really easy to fade out a screen in AMOS but without a long list of colours it is impossible to fade one in. This next program shows you how to fade in a screen using the unique Amiga screen cloning operation.

```
While Screen<>-1
Screen Close Screen
Wend
Do
Show On
F$=Fsel$("","","")
Hide On
If F$<>""
NICEIFF[F$]
While(Mouse Key=0)           ➡
and(Inkey$="")
Wend
Fade 2
Wait 24
End If
Loop
Procedure NICEIFF[A$]
Auto View Off
Load Iff A$,0
Screen Clone 1
Screen To Front 0
For X=0 To 31
Colour X,0
Next X
View
Auto View On
Fade 2 To 1
Screen Close 1
Wait 24
End Proc
```

AMOS GOES LOOPY!

BY LEO DOUGLAS

Many of the tasks that a computer excels at are the repetitive operations such as mathematical calculations and data handling. For the repeated processing of these tasks, programmers structure their code in loops. Loops are used to repeat certain sections of a program when required, with each repetition of a loop being referred to as a PASS. AMOS uses 4 different loop structures, and each has its own appropriate uses. To illustrate the form that these various loop structures take, I have written a simple routine in each one which counts up from 1 to 10, printing each pass number as it is reached. You may wish to type the examples (in bold text) in the AMOS editor and run them to see how they work, and the similarities and differences between them.

TYPES OF LOOP

INCREMENTAL LOOPS are carried out a specific number of times and terminate when the final value is reached.

FOR....NEXT Loop

The For-Next loop is always repeated a fixed number of times as defined by a variable in the first line of the loop. The point at which the loop ends must always be known before it is executed. The structure of the For-Next loop is as follows:

FOR <variable name>=<starting value> TO <end value>
<Do these commands>
NEXT <variable name>

For example, the following loop repeats itself 9 times with the value of A being incremented by 1 each time the command NEXT A is reached, and it ends when A becomes equal to 10.

```
FOR A=1 TO 10
PRINT "This is pass number ";A
NEXT A
```

The For-Next loop is most useful for operations which involve the same piece of code being repeated for a fixed number of times, such as reading data into or out of an array.

ITERATIVE LOOPS are repeated until a given condition is met, and will continue indefinitely, if necessary, until it is satisfied.

DO....LOOP Loop

The Do-Loop structure is unique in that it is purposely designed to repeat the commands between the DO and LOOP statements infinitely. To leave a Do-Loop loop, an EXIT or EXIT IF statement must be used within the loop to jump out when a required condition is met. A general Do-Loop has the following format:

DO
<These commands>
EXIT IF <condition is met>
LOOP

The Do-Loop form for our counting routine is:

```
DO
A=A+1
```

```
PRINT "This is pass number ";A
EXIT IF A=10
LOOP
```

The Do-Loop structure is slightly more flexible than the other forms of loop because the EXITing condition can be checked for and acted upon anywhere within the Do-Loop. A Do-Loop can be used for the main program loop, containing all of the major program routines and EXIT and/or EXIT IF statements to terminate the loop on successful completion of the program.

REPEAT....UNTIL Loop
The Repeat-Until loop will, not surprisingly, repeat a given list of commands until a condition, or series of conditions, becomes true. The basic form of a Repeat-Until loop is:

```
REPEAT
<These commands>
UNTIL <condition is met>
```

The Repeat-Until loop is strictly an iterative loop, but it can be made into a (somewhat crude) form of the For-Next loop with the following structure:

```
REPEAT
A=A+1
PRINT "This is pass number";A
UNTIL A = 10
```

This use of the Repeat-Until loop is not recommended but serves to illustrate the similarity in which the computer processes all such loops.
The Repeat-Until loop structure is best used for sections of code which will always need to be executed at least once, but may need to be repeatedly carried out; such as menus, data sorting routines or even data input, where the loop can be written to repeat itself until the user is happy that the information he or she has entered is accurate.

WHILE....WEND Loop
The While-Wend loop is the opposite form of the Repeat-Until loop; its first line specifying the requirements which must exist in order that the loop may be executed. The loop is then repeated until those conditions are no longer true, when it terminates at the WEND instruction. While-Wend loops have the following general structure:

```
WHILE <condition is not met>
<Do these commands>
WEND
```

```
WHILE A<10
A=A+1
PRINT "This is pass number ";A
WEND
```

The fundamental difference between the Repeat-Until and the While-Wend, is that the former loop will ALWAYS be executed at least once, but the latter may never be executed at all. This is because of the position of the statement which checks whether conditions to stop the loop have been reached. With the Repeat-Until loop structure, conditions are not checked until the last line (UNTIL ....) and so the preceding commands will always be executed just beforehand, even if the value(s) required to end the loop is true. The While-Wend loop, however, examines those conditions BEFORE it carries out any part of the loop (WHILE....) and if they are true it will not execute the loop at all.

LOOPS WITHIN LOOPS....

Loops can be placed within loops if necessary, in a process called nesting. Below is an example of a For-Next loop nested within a Repeat-Until loop:

```
REPEAT
FOR A=1 TO 10
PRINT "This is pass number";A
NEXT A
PRINT
UNTIL MOUSE KEY
```

The For-Next section is executed ten times, but the Repeat-Until loop surrounding it will continuously repeat the For-Next loop (and any other instructions inserted between the REPEAT instruction and the UNTIL instruction) until a mouse key is pressed.

Complex nesting of loops is not a job for the faint-hearted as it can get veerrry ("Been at the vodka again?" - Ed.) confusing trying to find errors in a piece of code which contains dozens of loops within loops (and believe me I know)!!

WHERE ARE WE?

If you need to contact us for any reason, we can be found somewhere between the tar pits and the marshes at:

36 CLEVERLY ESTATE
WORMHOLT ROAD
LONDON W12 OLX
ENGLAND

SOMEWHERE OVER.....

The rainbow command gives you some really fantastic opportunities to produce colourful displays without using lots of memory. This demonstration shows that it is possible to achieve fantastic looking software even on screens that would only normally allow you to display 2 colours. In fact this program does not even open a screen.

```
While Screen<>-1
Screen Close Screen
Wend
Set Rainbow 2,0,100,"","",""
Rainbow 2,0,150,100
TEMP=98
For LOP=1 To 32
Read C,A
For LOP1=1 To A
Rain(2,TEMP)=C
Dec TEMP
Wait 2
View
Next LOP1
Next LOP
GREEN_BIT:
Data $D0,12,$A0,1,$F0,1,$C0, ➡
11,$90,1,$E0,1,$B0,10,$80,1,$D0,1
Data $A0,9,$80,1,$C0,1,$90, ➡
7,$70,1,$B0,1,$80,6,$60,1,$A0,1
Data $80,5,$50,1,$90,1,$70,    ➡
4,$40,1,$80,1,$60,3,$30,1,$80,1
Data $50,2,$20,1,$70,1,$40, ➡
1,$20,1
```

FIRST ISSUE COMPETITION: ANSWERS AND WINNERS

The answers to last issue's competition were:

1 Richard Vanner once wrote for **ATARI USER**

2 The developers of the very first version of BASIC were
T. E. KURTZ and **J. G. KEMENY**

3 Peter Hickman began his programming days on the **SINCLAIR ZX81**

The first six correct answers drawn out of the "ALL ABOUT AMOS" lucky bucket were:
Simon Beales, T.S. Greenwood and **Mark Milner** who all win a copy of the AMOS COMPILER.
J. Chantler, Peter Cunningham and **Wayne Waite** who will each receive a copy of AMOS 3D.

Congrats to the winners and commiserations to everyone else, but if you didn't win, never fear: A new competition is wandering about just below!

COMPETITION: WIN COPIES OF FUN SCHOOL 4!

This issues competition is another challenge for super eggheads! Sort of. Just write the answers to the questions below (all of which can be found in this very magazine!) on a postcard, sealed down envelope, or crisp £50 note (just joking about the last one ("No we're not!" - Ed.)). Then post it to us before the closing date of **25th January 1992** and you could be the proud owner of a copy of **FUN SCHOOL 4** for the Under 5s, 5 to 7s or Over 7s.
QUESTION 1:
How many colours can a screen with one Bitplane display?
QUESTION 2:
Why was François Lionet seen "attacking" Richard Vanner with a briefcase at the World of Commodore Show?
QUESTION 3:
Name one of the commands listed in the AMOS manual which does not, in fact, exist.